**An Approach to Distributed Sensing in a Virtual Fishtank**

by

Mark L. Neri

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 21, 1997





Author________________ 5/21/97

Department of Electrical Engineering and Computer Science

Certified by________________

Bruce Blumberg

Thesis Supervisor

Accepted by________________

F.R. Morgenthaler

Chariman, Department Committee on Graduate Thesis

An Approach to Distributed Sensing in a Virtual Fishtank

by

Mark L. Neri

Submitted to the
Department of Electrical Engineering and Computer Science

May 21, 1997

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
And Master of Engineering in Electrical Engineering and Computer Science

# ABSTRACT

The simulation of many simple creatures whose interactions lead to complex global phenomena is of great interest to many people. Because simulations of this type are often computationally intensive, it is often useful to distribute the simulation across several computers connected by a network. A distributed octree data structure achieves both rapid computation of the creatures' actions as well as distribution of these computations among several computers. This approach exploits many of the constraints on creatures with limited visual ranges in a three dimensional space in order to allow simulation of many creatures in real time. This work will eventually lead to the construction of a Virtual Fishtank that will allow users to experiment with the phenomena that arise from the behaviors of many simple, interacting creatures.

Thesis Supervisor: Bruce Blumberg
Title: Assistant Professor of Media Arts & Sciences

# Table of Contents

# 1. Introduction

The Virtual Fishtank Project is a joint effort of the MIT Media Lab and the Boston Computer Museum. The goal is to construct an exhibit that will allow museum visitors to experiment with the behaviors that emerge from the interactions of large numbers of creatures that follow simple rules based on local information. The central element of this exhibit will be a large video display on which the interacting creatures will be displayed. Several separate computers will project this display. Various types of creatures will live in the tank, although the most prominent will be fish. Museum visitors will control these creatures in a variety of ways ranging from direct control to a powerful graphical programming interface.

The fishtank will provide an interesting environment with which creatures will interact. It will have walls and other obstructions, which will constrain the movement of the creatures. It will provide other environmental factors such as lighting and temperature. Also, there will be ways for visitors to interact directly with the creatures in the tank by feeding them or tapping of the glass of the tank, for example.

The goal of the exhibit is to show museum visitors that many phenomena that appear to be organized on a global scale actually arise from many simple agents following simple rules. These phenomena occur in a wide range of fields, from social animal behavior, to economics, to human group behavior. The Virtual Fishtank will play an essential role in communicating this idea of emergent behavior.

## 1.1. The Creatures

The most important elements of the Virtual Fishtank are the creatures that interact within it. Several types of creatures will populate the tank. Each will demonstrate a different type of interaction that leads to an interesting global effect.

One of the most prevalent creatures will be fish. These fish will not necessarily interact in the ways that real fish do, but they will exhibit interesting emergent behavior. The most obvious interaction among these fish will be the formation of schools, although other behaviors may be discovered as well. To help museum visitors to grasp the theme of the exhibit, they will be able to manipulate the behaviors of the fish in several different ways.

In order to provide a meaningful experience to a wide range of museum-goers, the fish will be programmable at a wide range of complexity levels. At the simplest level, visitors will be presented with sensory information for a single fish and will directly control that fish's movements. At a slightly higher level, they will be able to control the behavior of groups of fish by manipulating a few simple parameters. Finally, a graphical programming interface will allow visitors to construct a wide range of fish behaviors. The results of changes made to a fish's behavior will be immediately reflected in the behavior of that fish on the large displays.

Although the fish will be the focal point of the exhibit, other creatures will play a crucial role in communicating the overall message. The Epistemology and Learning group at the MIT Media Lab has been experimenting with behaviors that emerge in a wide variety of systems ***(*****Resnick94a*****)*** ***(*****Resnick94b*****)***. Many of these will be represented by other types of creatures in the tank. The types of interactions between these creatures will each be unique, but there will be some common theme among all of them which will help to communicate the theme of the exhibit. Museum-goers will learn that all of the system behaviors are the result of individual creatures following simple rules and using only local information.

## 2. The Problem

Many problems will have to be solved before the Virtual Fishtank can be put into operation. Simple yet effective user interfaces will have to be constructed that allow museum visitors to manipulate the behaviors of creatures in the Fishtank. Models of the creatures in the tank will have to be built that include both the creature's structure and it characteristic movements. Behaviors will have to be designed that users can use as a starting point for their own exploration. Most importantly, the simulator itself will have to be constructed.

This thesis describes the implementation of a prototype for this simulator. Specifically, this thesis presents a solution to the problem of rapidly finding the objects that a creature can see, using this information to decide what movements to make, and finally, making these movements. To meet the computational needs of this project, these processes harness the power of several computers connected together with a network. For testing and demonstration purposes, models of the motions and behaviors of schooling fish are implemented as well.

### 2.1. The Distributed Octree

The major part of this thesis focuses on the design and implementation of a distributed data structure that will be able to simulate the behaviors of creatures in the tank. This data structure must allow rapid computation of the sensory inputs that a creature will use to decide how to move around in the tank. It also must be arranged so that it can run in real-time, despite the latencies involved in network communication. An octree is used to achieve these goals.

The octree allows a creature to find other creatures that are within its field of view very quickly. Different creatures in the tank have different fields of view, which depend primarily on the placement of the creature's eyes. A typical fish can see roughly ninety degrees up and down and has about a three-hundred-degree field of view from left to right. Crabs can see the entire space that is above them, but nothing below. Further, the distance that a creature can see is limited.

Two important techniques facilitate rapid location of the creatures that are within a creature's field of view. The first is the precomputation of octrees that represent sub-sets of the space within the main octree. For each leaf node and for each possible viewing distance, an octree is computed that represents only the regions of space that could be seen. These pruned octrees are used as a starting point for the visual computation. The second technique is the incremental computation of information about the relationship between a node in the octree and the planes that define the shape of a creature's field of view. By using information about this relationship for a parent node, a great deal of computation is saved when recursing to a child node.

The distribution of the simulation between several computers connected by a network is achieved by having each computer simulate a particular region of space. These regions of space correspond to branches of the octree. Because each creature's range of vision is small in comparison to the size of the simulation space, most the information on which it bases its behavior is computed on the same computer on which it is computed. This means that the amount of information that needs to be transmitted across the network is small. In order to maintain this correspondence between a creature's location and the computer on which it is simulated, control of creatures must be shifted among computers as creatures move around the tank.

# 3. Implementation

The prototype fishtank developed many of the principles of that will be needed to construct the final version of the Virtual Fishtank museum exhibit. The central component is the distributed octree, which achieves rapid computation of creatures' fields of view. It also allows distribution of the simulation across a network of computers connected by a network.

The prototype was implemented in the Java programming language. Java was chosen because the platform on which the final simulation will run was unknown when the program was written. The program uses a very object oriented approach which will make it easy to port to the program to a compiled language like C++ if the Java implementation is not fast enough. The major drawback to using Java is that its performance does not measure up to compiled languages. Still, the simplicity and portability of the language make it quite suitable for building a prototype.

A number of additional components were necessary in order to test and demonstrate the simulation. Microsoft's Direct3D rendering library was chosen to handle the rendering needs of the project. Sample behaviors and movement constraints were based on a combination of previous work ***(*Tu94*)*** and observations made at the New England Aquarium. To allow user's to program the fish interactively, a visual-programming tool was interfaced to the simulator ***(*Travers94*)***.

## 3.1. The Distributed Octree

The heart of the fishtank simulator is the octree data structure. This structure partitions the space in which the tank's inhabitants interact with each other.

The top node of the octree represents a cube with edges of length two, centered at the origin. This cube is recursively divided into octants. This division is performed a static, configurable number of times. The optimum number of divisions depends on the number of creatures that are to be simulated. For several dozen creatures, a three-level tree with sixty-four leaf nodes was found to give the best performance.

The creatures that are in the tank are stored in the leaf nodes of the octree. Each creature is represented by a single point. Each creature is stored in the node that represents the region of space in which the creature's representative point lies. When a creature is inserted into the tank, the tree is recursed from the top down until the correct leaf node is found. At each time step, when a creature moves, the simulator checks if the creature has left the node in which it was on the last time-step. If it is not, the tree is searched upwards until a node that contains the creature's new position is found, and then it is recursed downward to the leaf node where the creature should be stored is found. In this way, every creature is kept in the correct node of the octree.

### 3.1.1. View Computation

A significant challenge in the design of the fishtank was to find a way of representing the data collected by the creatures' visual senses. This representation had to capture enough information for the creature to base interesting behaviors on, yet it had to be simple enough to make behavioral programming easy. The method chosen was to use structures called accumulators, which aggregate certain properties of a group of creatures in the tank and predicates, which test some property of a creature. An accumulator and a predicate can be combined to form a structure called a view-result. Each behavior defines a list of view-results that it needs.

When this visual information is needed to process a creature's behavior, the simulator finds every object in the behaving creature's field of view. For each view-result that the behaving creature needs, every visible object is tested against the predicate of the view-result. If the observed object matches the predicate, the accumulator is applied to that object. After this process, the state of the accumulators in the view-results represents information about the objects within the creature's field of view. This information is then used to decide how the creature should behave.

A number of different predicates were constructed to create demonstration behaviors for the fishtank simulator. One could test whether a creature was of a specific color, or if it was of the same color as the observing creature. Another could test if one creature was within a specified distance of another. Also, predicates could be combined or inverted using Boolean operators. Together, these predicates were able to represent a variety of conditions.

A few useful accumulators were also written to facilitate the programming of interesting behaviors. One could find the creature that was closest to another. Another could find the centroid of a set of creatures. By combining these with various predicates, many different bits of simple, useful information about the objects around a creature could be computed in a way that was easy for a programmed behavior to work with.

The computational challenge was to find the find all of the other creatures within a given creature's field of view so that accumulator-predicate pairs could be applied to them. A careful choice of the exact shape of the field of view was represented was critical for this. Many useful fields of view can be represented by the result of boolean operations applied to a sphere and small number of half-spaces whose defining plane passes through the center of the sphere. For simulating the field of view of a fish, the shape that is needed is a sphere with a slice of approximately sixty degrees taken out of it. This represents a three-hundred-degree field of view as well as the limited range of vision that is possible underwater.

Using this representation of a field of view, the visual computation process can be broken into two sub-problems. The first is to find the creatures that lie within the sphere or more simply, to find the creatures within a given distance of the viewing creature. A pruned octree data structure was used to achieve this. The second is to find the creatures that are within the correct set of half-spaces. This was done using structures called view distances. By carefully interleaving these two techniques, the time need to compute visual data is minimized.

#### ***3.1.1.1. Pruned Octrees***

The pruned octree structure that is used for visual computations represents a subset of the space that lies within the complete octree. Each pruned octree has a leaf node, referred to as the base of this pruned tree, and a specific distance associated with it. The pruned tree contains all of the leaf nodes in the main octree that contain some space that is within the specified distance of the base node. Also, a flag within each node of the pruned octree is set to indicate whether all space in the node lies within the specified distance of all space within the base node. In order to keep the size of the data structure as small as possible, the recursive expansion of a node is performed selectively. If either all of node's space is within the specified distance of the all of the base node's space, or no point in the node is within the specified distance of any point in the base node's space, the node is not expanded.

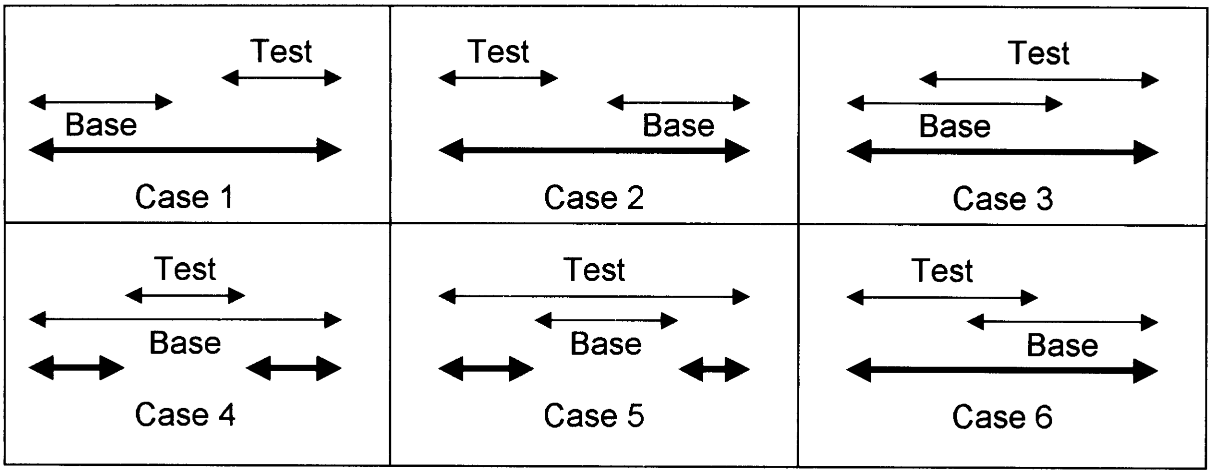


***Figure 3.1-1 The six possible relationships between two intervals and their maximum distances.***

*The six cases shown correspond to the cases six cases described in Listing 3.1-1. The top line represents the test segment, while the middle line represents the base segment. The thick, lower line indicated the maximum distance between any two points that lie within the intervals. In cases 4 and 5, the maximum distance is the length of the longer indicated segment.*

When the simulation is started, many pruned trees are generated. A set of several discrete distances is build into the program. At each node, a pruned octree is constructed using the node as the base node for each of the discrete distances. The storage of these pruned trees can be significant, but is not unreasonable, especially for octrees that have a depth of four or less.

One tricky aspect of the computation of the pruned octrees is to determine when all space within one octnode is completely within a certain distance of all space within another. This is done by comparing the specified distance to the maximum distance between points in the octnodes. This problem can be simplified by solving the problem in each of three dimensions and then combining these results using a simple distance formula. This is possible because the nodes are axis-aligned cubes.

The problem that must be solved in one dimension is to find the maximum distance between a base interval and a test interval, given the minimum and maximum point of each interval. Each of the possible relationships between these two intervals falls into one of six cases, as described by Figure 3.1-1 and Listing 3.1-1.

***Listing 3.1-1 Computing the maximum distances between two intervals***

```
double
MaxDist (double baseMin, double baseMax, double testMin, double testMax)
{
    //Check if Base is completely to the left of Test
    if (testMin > baseMax)
        //Case 1
        return testMax - baseMin;

    //Check if Base is completely to the right of Test
    if (baseMin > testMax)
        //Case 2
        return baseMax - testMin;

    //Check if the minimum of Base is in Test
    if (baseMin < testMin)
    {
        //Check if Test is completely in Base
        if (baseMax < testMax)
            //Case 3
            return testMax - baseMin;
        else
            //Case 4
            return Math.max(testMin - baseMin, baseMax - testMax);
    }
    else
    {
        //Check if Base is completely inside of Test
        if (baseMax < testMax)
            //Case 5
            return Math.max(baseMin - testMin, baseMax - testMax);
        else
            //Case 6
            return baseMax - testMax;
    }
}
```

Similar subtleties arise in finding the minimum distance between the any point in a test octnode and any point in another octnode. Again, solving in each dimension and combining the results using a simple distance equation can solve this problem. The algorithm in Listing 3.1-2 performs this computation.

***Listing 3.1-2 Computing the maximum distances between two intervals***

```
double
MinDist(double baseMin, double baseMax, double testMin, double testMax)
{
   //Check if Base is completely to the left of Test
   if (testMin > baseMax)
      return testMin - baseMax;

   //Check if Base is completely to the left of Test
   if (baseMin > testMax)
      return baseMin - testMax;

   //The intervals overlap
   return 0;
}
```

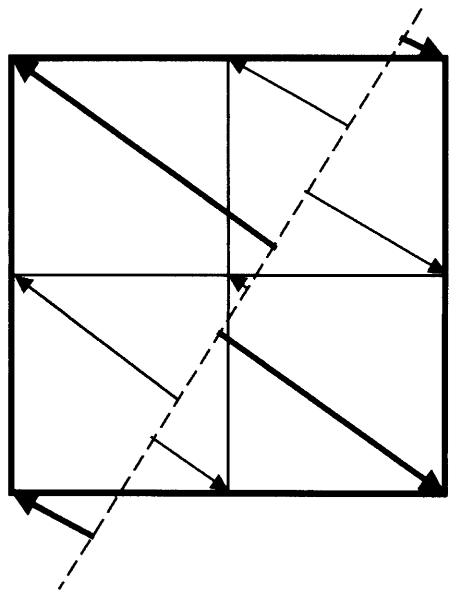

***Figure 3.1-2 - Computing signed distances***

*Given the signed distances from a line to the corners of a square, it is very easy to compute the signed distances to the corners of the square. The distances to the midpoints of the edges of the square can be found by averaging the endpoints of the edges. The distance to the center of the square can be computed by averaging the distances of the corners of the square.*

#### ***3.1.1.2. View Distances***

The view distances data structure represents the relationship between the half-spaces that intersect with a sphere to define a view and the nodes of an octree. A view distance is represented by group of other structures called plane distances.

The plane distance structure represents the signed distances from the corners of a node in the octree to a specific plane. There exists a relationship between the signed distances of the corners of a node in the octree and the signed distances of the corners of its child nodes. This relationship allows the distances of the corners of a cube's octants to be computed rapidly based on the distances of the cube's corners. There are twenty-seven distinct corners of the octants of a cube. Eight of these corners correspond to the corners of the cube, so their distances from the plane are drawn directly from the cube. Twelve octant corners lie on edges of the cube. Their distances can be computed simply be averaging the distances of the endpoints of the edge on which they lie. Six more corners lie in the center of faces. The distances to these can be computed by averaging the distances form two diagonal corners of the respective face. Finally, there is one point in the center of the cube whose distances can be computed by averaging the distances of two corners on a main diagonal of the cube. Figure 3.1-2 shows these relationships in two dimensions.

These plane distances can be used to compute very quickly whether all, some, or none of the space in an octnode lies within the half space defined by the view plane. This is done by finding the corners of the top node of the octree that have highest and lowest distance from the plane. If the minimum distance is positive, then the octnode lies completely on one side of the plane. If the maximum distance in negative, then the entire octnode lies on the other side of the plane. If neither of the above is true, then the octnode and the plane intersect.

Computation time can be decreased further by recognizing that the minimum and maximum distances on nodes within the tree can be found one the same relative corners as the minimum and maximum distances on the root node of the tree. This means that once the signed distances need only be searched once, at the top of the tree, in order to find the positions of the minimum and maximum plane distance.

These plane distance structures are combined to form view distance structures that can represent more complex shapes, such as a fish's three-hundred-degree field of view. Plane distance structures can be combined into either intersections or unions of the spaces the individual planes represent. If an octnode is completely within the half-space defined by any of the planes in the union, it is completely within the union. If it is partially within any of the half-spaces, it is partially within the union. Only if a node is completely outside of each of the half-spaces is it completely outside of the union. Similar reasoning can be applied to implement the intersection operation.

### *3.1.1.3. Computational Process*

In order to find the creatures that another creature can see, the pruned-octree technique and the view distances technique must be carefully interleaved. This allows the branches of the octree that do not contain any space that is within the creature's view to be passed over quickly.

The process begins by finding the appropriate pruned octree, which has already been generated when the simulation began. Each creature stores the node of the main octree that it is currently in. Each node of the main octree stores several pruned octrees, each of which represents the regions of the main octree that is within a certain distance of it. So, when a creature needs to find the objects that are within its field of view, it finds the top of the pruned octree that corresponds to the distance that creature can see.

Once the correct pruned tree has been found, it is recursed by the code in Listing 3.1-3. This procedure takes a list of accumulator-wildcard pairs, a field of view, and a view distances structure as described above. This original view distance structure is computed by taking the standard formula for the distance from a point to a plane. This code first checks to see if the node in the main octree that corresponds to the current node of the pruned octree is empty. This check is very important for performance because it avoids a great deal of processing in branches of the octree that contain no objects.

If the current node of the pruned tree corresponds to a leaf of the main octree, the accumulator-wildcard pairs are applied to the objects within that node. There are two ways that this application can be performed. If the precomputation of this pruned octree found that all of its space was completely within the specified distance of the node that the test creature is in, the application is performed to all objects that are within the half-spaces that define the view. Otherwise, the objects in the leaf node are tested to determine if they are in the specified distance of the test creature as well as if they are in the half-spaces that define the creature's view before the application is performed.

If the current node of the pruned octree is not a leaf node, then each of its child nodes that is in the pruned tree is processed in turn. First, a view distances structure for the child node is computed based on the view heights for the current node. If this structure indicates that the child node is completely outside of the field of view, processing continues with the next child. If the child node is partially within the field of view, the function is called recursively to subdivide the node. If the child node is completely within the field of view, it is recursed using a different procedure that does not perform any further view calculations but still does distance comparisons. This code is shown in Listing 3.1-4.

### *Listing 3.1-3 Interleaving of Pruned Octree and View-Distances Techniques*

*The PartialLook function is called to find the objects within a creature's field of view.*

```
void
PartialLook(CLookFor lookfor, CView view, CViewDistances viewDistances)
{
    if (Octree.NumObjects == 0)
        return;

    if (Octree.IsLeaf)
    {
        if (CompletelyWithin)
            ((COctreeLeaf)Octree).AccumulateInView(lookfor, view);
        else
            ((COctreeLeaf)Octree).AccumulateInViewDistance(lookfor, view);
    }
    else
    {
        CViewHeights childDistances = view.GetEmptyHeights();

        for (i=0; i<8; i++)
        {
            //Check if the child node is in the pruned octree
            if (Children[i] != null)
            {
                //Compute the view distance of the child node based on
                //the view distances of this one
                view.GetOctantHeights(childDistances, i, viewDistances);

                //Check if the child is not completely out of the view
                if (!view.OctantOutside(childDistances))
                {
                    //Check if the child is completely within the view
                    if (view.OctantInside(childDistances))
                        Children[i].CompleteLook(lookfor, view);
                    else
                        Children[i].PartialLook(lookfor, view, ChildHeights);
                }
            }
        }
    }
}
```

***Listing 3.1-4 View Computation for a Node Completely Inside of the View-Planes***

*For efficiency, branches of the octree can be recursed with CompleteLook when it is known that the top node of the branch is completely inside of the view planes. This avoids view-plane calculations for all descendants of this node.*

```
void
CompleteLook(CLookFor lf, CView v)
{
        if (Octree.NumObjectss == 0)
            return;

        if (this.CompletelyWithin)
            Octree.AccumulateAll(lf);

        if (Octree.IsLeaf)
        {
            ((COctreeLeaf)Octree).AccumulateInDist(lf, v);
        }
        else
        {
            for (int i=0; i<8; i++)
            {
                //Recurse any children that are partialy in the distance
                if (Children[i] != null)
                    Children[i].CompleteLook(lf, v);
            }
        }
    }
}
```

### 3.1.2. Network Communication

In order to simulate the number of creatures that the Virtual Fishtank will contain it is necessary to harness the power of several computers. The octree structure lends itself well to the distribution of the simulation of the creature's interactions. The key to making this distribution effective was to limit the amount of information that had to be transmitted from one computer to another. One of the central ideas of the study of emergent behavior is that the interacting creatures have only local information about their environment. If network communication is to be minimized, the information that a creature needs to execute its behavior should be computed on the same computer that the on which the creature is simulated. This led to the conclusion that each computer that runs the simulation should take care of simulating the behaviors of the creatures in one region of space. Because each branch of the octree represents a single region of space, it seemed reasonable to assign different branches of the octree to different computers.

The computers used to simulate the Virtual Fishtank were connected using a standard Ethernet network, using the TCP/IP protocol. The standard Java classes were used to interface the software to this network.

#### *3.1.2.1. Establishing Connections*

In order to make several computers work together to simulate the creatures in the Virtual Fishtank, a number of configuration and setup activities have to take place before the simulation can begin. The computers must agree on which systems will simulate which regions of space. Also, each computer must locate and establish communications with the other computers that it must exchange information with during the simulation.

The decision about which computers will simulate which parts of space is left to the user of Virtual Fishtank. The depth of the octree along with a listing of which computers should control which nodes of the tree is read in from a file when the simulation starts. The computers for each leaf of the tree are specified with an index value. These values are mapped to IP addresses and ports through another file. This allows the system to be easily reconfigured for the purpose of experimentation. The configuration files are shared between computers using a standard file sharing system in order to ensure that all of the computers running the simulation have the same configuration information.

After reading in the configuration files, each computer determines with which other computers it must communicate with in order to get the information it needs to simulate the behaviors of the creatures in the part of the tank that it controls. It does this by looking at the pruned octrees that correspond to the largest viewing distance that a creature in the tank may have. This value was chosen to be one quarter of a unit, which is one eighth of the linear dimension of the fishtank. Then, for every leaf node in the octree that the configuration files specify should be simulated on that computer, the simulator notes that it needs to get data for the leaves in the largest pruned octree of that leaf. It also determines, from data read from the configuration files, from which computer it should get this data. By exploiting the symmetry of the octree and the pruned octrees, each computer also computes, for each node that it is to simulate, which other systems will need data about the creatures stored in that node.

After the simulator has determined which data it will need to simulate the behaviors of the creatures in the space that it is to control as well as the computers from which this data should come, it needs to establish TCP connections with these systems. It also needs to allow these systems to establish communication with it. An important feature of these connections is that they are symmetric, that is, every system that a system connects to will try to connect to that system. Therefore, each system first sets up a server port for other systems to connect to. Then, it tries to connect to each of the systems that has data that it needs. It continues these attempts until it has successfully connected to every other computer that has data it needs and these computers have connected to it as well.

#### *3.1.2.2. Transmission of Visible Data*

Once all of the needed network connections have been established, the simulation can begin. Initially, on each system, all octree leaves that are not to be simulated on that system are set to contain no objects. Any creatures that are initially in the tank are only stored in the computer that simulates the octree leaf in which those creatures are initially placed. At this point, the simulation of the first time step can begin.

To simulate a time step, each computer running the fishtank simulator performs the behavior defined for each creature that is a leaf of the octree that it is assigned to simulate. After the behavior of each creature is simulated, information about the new values of its observable properties is sent across the network to all of the other computers that need it. This information includes of the creature's position, direction, orientation, speed, size, shape, and an index indicating which leaf node the creature is in after the time-step. After all of the creatures in an octree leaf have been processed, a special code is sent across the network to indicate that the data for all of the creatures in the current leaf of the octree has been sent. Once all local leaf nodes have been processed, the simulation pauses until it has received the information from other computers about the objects in nearby leaf nodes.

While a computer is simulating the behaviors of the creatures in its region of space and sending information to other computer about these creatures, it must also receive and process data that is being sent to it from other systems. When data is received from the network, the first field indicates which leaf of the octree the object represented by this data should be put into. In order to avoid threading problems, the creature is not immediately added to the list of objects in the specified leaf of the octree. Instead, this information is added to a temporary list within that node. This process continues until all data has been received. This can be determined by counting the number end-of-node codes that are received and comparing the result to the number of nodes that the initialization code found were needed from other systems.

Once all data from the network has been received and the simulation of creature behaviors has finished, each computer must prepare the data from the network for use in the next time step. In leaf nodes that are not simulated locally, this is achieved by replacing the old list of objects in the node with the list of objects received from the network. If the leaf node is to be simulated locally, then the objects in the list that was received from the network are added to the list of objects already in that node. Once this is done, the fishtank is rendered and the next simulation step is performed.

#### ***3.1.2.3. Transfer of Control***

The system described above will cause the simulation processing for a creature to move among computers as the creature's behavior takes it to different areas of the fishtank. One small modification must be made to allow this. When the information for a creature is sent across the network, it must be determined whether the leaf of the octree that the creature's new position lies within in being simulated locally. If it is not, then the information needed to simulate the creature's behavior must be sent across the network along with the observable information for that creature. This additional information includes the type of body and behavior that define the creature as well as any parameters of it body or behavior. When a system receives this additional data, it can continue the simulation of the creature's behavior seamlessly.

## 3.2. Auxiliary Components

In order to test and demonstrate the techniques described above, several auxiliary components had to be constructed. Creatures with interesting behaviors and believable movements had to be created. For this purpose, schooling fish were used. The behaviors that lead to this behavior and the constraints on the motion of fish have been well explored, so it was possible to distinguish between programmed behaviors and artifacts of implementation errors. Also, in order to see what was going on in the tank, a rendering system had to be linked with the simulation.

### 3.2.1. Behaviors

The behaviors that lead to believable fish schooling are described in ***(Tu94)***. There are three important parts of this behavior. The first is that fish should go towards the centroid of the fish they are to school with. They must also go in the same direction as fish that are close to them. Finally, they should go away from other fish that are very close to them. Obviously, these rules are often contradictory.

To deal with this, a weighting scheme was used to reconcile these conflicts among rules. The goal of this scheme is to determine a direction and a speed that the fish will seek to attain. This desired speed is simply the weighted average of the speeds suggested by the three rules above. The desired direction is represented by a vector, which is computed by taking a weighted average in each dimension of the directions indicated by the rules.

There is a wide range of weights that will lead to convincing schooling behavior. Generally, turning away from other fish that are too close has the strongest weight. Going in the same directions as a nearby fish has a somewhat lower weight. Turning towards the centroid of the school has an even lower weight. Varying the weights changes the character of the school. If turning away has a very high weight, the fish scatter and do not seem like a school at all. If going in the same direction has a high weight, the fish swim parallel to each other, but do not come together. Increasing the weight of the centroid rule leads to a group behavior that is more like a swarm than a school. The schooling behavior was improved further by using non-constant weights for the different rules. For example, the weight for the turning away behavior was scaled by the reciprocal of the distance to the other fish. The most interesting combination of weights is a matter of taste and is best obtained through experimentation with the Virtual Fishtank.

One of the most challenging aspects of programming the fish behaviors was making them avoid the walls of the tank. It is straightforward to find the nearest wall and to figure out which direction to go to avoid this wall. The difficulty arises in determining how to combine wall avoidance with the other rules that lead to schooling. Although there is quite a bit of flexibility in the three schooling rules, when the wall avoidance rule is added, it becomes significantly more difficult to balance the two behaviors. Through careful tuning and experimentation, however, a weighting mechanism was devised that led to fish that formed schools, but did not run into the walls very often.

### 3.2.2. Bodies

In order to make fish move around the fishtank in a believable way, it was necessary to constrain their movements. Based on previous work ***(Tu94)*** as well as observations made at the New England Aquarium, a simple model of how fish are able to move around a fishtank was constructed. There are three major constraints in this model. The rate at which a fish can turn from left to right (yaw) is limited. The rate at which a fish can turn up and down is also limited (pitch). Finally, the maximum angle at which a fish can swim up or down is limited. Although the actual constraints on fish are substantially more complex than these three constraints, this model did lead the fish to move around the tank in believable ways.

Given this model of the constraints of fish motion, a method of translating the desired movements specified by the fish's behavior to actual fish movements that followed the constraints was needed. The problem is, given a desired speed, a desired direction, and actual speed, and an actual direction make the actual position and direction as close as possible to the desired. This problem must be solved without violating any of the movement constraints imposed by the model. This is done using basic linear interpolation for the speed and trigonometric interpolation for the direction. The boundary cases for the direction interpolation are somewhat troublesome. Listing 3.2-1 shows how they are handled.

***Listing 3.2-1 Interpolation of a Fish's Direction***

```
void
SeekDesiredMovement(double dTime)
{
   double tilt = Direction.GetTilt();
   double angle = Direction.GetAngle();

   double desiredTilt = DesiredDirection.GetTilt();
   double desiredAngle = DesiredDirection.GetAngle();

   //Make sure that the creature does not go past the maximum allowed tilt
   desiredTilt = Math.max(-MaxTilt, desiredTilt);
   desiredTilt = Math.min(MaxTilt, desiredTilt);

   //For robustness, make sure the creature has no exceeded the allowed tilt
   tilt = Math.max(-MaxTilt, tilt);
   tilt = Math.min(MaxTilt, tilt);

   //Decide whether to tilt up or down, but don't go too far
   if (tilt < desiredTilt)
      tilt = Math.min(desiredTilt, tilt + TiltRate * dTime);
   else
      tilt = Math.max(desiredTilt, tilt - TiltRate * dTime);

   //Put the angle between -pi/2 and pi/2
   if ((desiredAngle - angle) > Math.PI)
      desiredAngle = desiredAngle - 2*Math.PI;

   //Decide whether to go left or right, but don't go too far
   if (angle < desiredAngle)
      angle = Math.min(desiredAngle, angle + AngleRate * dTime);
   else
      angle = Math.max(desiredAngle, angle - AngleRate * dTime);

   //Set the fish's new direction
   Direction.Set(angle, tilt);
}
```

Another constraint of the fish in the tank that is different from the ones mentioned above is they cannot swim outside of the tank. It is especially important because, for efficiency reasons, it is assumed that all of the objects in the main octree have coordinates that lie inside of the tank. While a carefully programmed behavior should not cause a creature to bump into the walls of the tank, the fishtank is intended for experimentation, so this cannot be guaranteed. Therefore, it is necessary to check, after each movement, whether a creature has tried to go through the walls of the tank. If a creature has done this, then it is necessary to do something to the creature that will move it to a legal position. Also, the creature should be positioned such that it does not run into the wall again on the next time-step. When a fish runs into a wall, it is immediately rotated so that it faces direction away from the wall. This sometimes leads to unnatural movements, but less drastic steps seemed to lead to many situations in which creatures would repeatedly bump into the same wall. This was quite frustrating to behavior programmers.

### 3.2.3. Rendering

The rendering of the Virtual Fishtank prototype was done using Microsoft's Direct3D rendering library. This library was chosen because it was one of the only three dimensional rendering system that could be easily interfaced to a Java program. Also, with the recent introduction of hardware-based graphic accelerators for PC's, these systems are able to meet the rendering demands of the prototype fishtank. The most significant drawback of this choice was that the simulation could run only on the Windows 95 platform.

The three dimensional models that were used to render the fish were created using a high end graphics package. They were manually simplified and converted into models with dozens of faces that looked good, yet could be rendered in real-time. Several techniques were used to create a sense of depth in the fishtank. Texture backgrounds with perspective were used. Also, depth-cueing by shifting distant fish's colors toward the blue color of the water was quite effective.

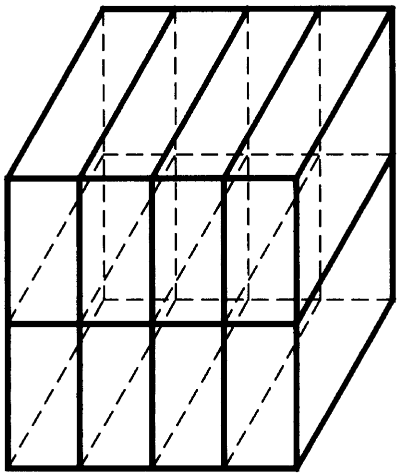

***Figure 4-1 – The Division of the Simulation Space Used For Testing***

*This shows the spaces that were assigned to the eight Virtual Fishtank simulators. Because only four computers were available for testing, two instances of the simulator were run on each one. Each computer ran the simulation for one vertical slab of the simulation space. The slabs were split horizontally, and each half was simulated by a separate instance of the Virtual Fishtank simulator.*

# 4. Results

The Virtual Fishtank simulation described has been tested using a network of four PC's. Each PC was a fast, Pentium-based machine with a Diamond Stealth 3000 video card running Windows. They were connected together with a 10Mb/s ethernet network. Two instances of the fishtank simulation program were executed on each machine, in order to simulate the way that the simulation would run on eight machines. The simulation space was divided into four slabs, each of which was simulated on one computer. Each slab was divided in half, and each half was simulated by a separate instance of the simulator as show if Figure 4-1.

Using this configuration, the behaviors of sixty-four schooling fish were simulated. The simulation ran nearly as fast on the four computers as a sixteen fish simulation ran on a single computer. The overall slowdown was approximately twenty-five percent. Some of this slowdown was due to the overhead involved in performing the network communication between nodes.

Additional slowdown came from the synchronous nature of the simulation. That is, one computer cannot begin simulating a given time-step until all of the computers in the simulation have finished the previous step. Therefore, one simulation step takes at least as long as the simulation of all the creatures being simulated by any one computer. The slowdown due to this effect increased when creatures with behaviors that caused them to all stay very close to each other. In this situation, all of the simulation is done on the single computer that controls the region of space in which the creatures are clustering.

If the number of computers available for running the simulation were increased, a few new problems could arise. First, the communication between the computers could start to exceed the amount of bandwidth available on the network. This problem would be especially bad if a network with a bus-configuration were used because each computer would need to perform more network communication, and there would be more computers communicating. Fortunately, for a given assignment of simulation space to computer's, the potential pairs of computers that need to communicate is fixed, so, either a segmented network or point to point links could be used to improve communication bandwidth.

Another problem that might be encountered if the simulation were run on many computers is that the amount of communication between systems could grow disproportionately larger. This could happen if the size of the space simulated by each computer became small in relation to the visual range of the creature's being simulated. This would be caused by the inevitable dependencies between the creature's behaviors. The only way to reduce these dependencies would be to decrease the visual range of the simulated creatures. This may be a reasonable thing to do, because when the number of fish that are being simulated is increased, the scale of the simulation space should be increased as well in order to avoid crowding. This increase in scale would correspond to a decrease in visual range, which would eliminate some of the inter-creature dependency.

The results of this project suggest that this approach is suitable in a variety of situations, despite its limitations. The techniques used to compute visual information using the octree structure would be useful in any situation where dozens of creatures are interact in a three dimensional space. To deal with more creatures, the depth of the octree could be increased in order to improve the efficiency of algorithms. With fewer creatures, the depth could be decreased, although with less that ten, a simpler data structure would be more appropriate.

The distribution of the octree is also appropriate in a range of situations. Using the spatial structure of the simulation environment to allocation computational effort is most effective when creatures tend to be distributed evenly within the simulation space. Ideally, at any time, the same number of creatures would be in the space simulated by each computer. This was not the case with some schools of fish. When fish were programmed to form dense schools, the performance of the simulator decreased significantly. A possible way to work around this problem would be to have each computer simulate several small, far apart regions of space. This would increase network demands, but would lead to a more even distribution of computation.

# 5. Previous Work

Much work has been done that will be useful in the implementation of the Virtual Fishtank, although many areas seem to be unexplored. Two areas of research were relevant to the implementation of the Virtual Fishtank. First, many ways of specifying the types of simple behaviors that will lead to interesting global phenomena have been explored. Ideas from these works were used to determine what sensory inputs would be needed, how these inputs should be processed, and what range of actions that creatures should be able to take. The second area of interest is the distribution of the Fishtank simulation. Some general principles are found in work on generic distributed simulation. This work, however, does not cover many of the details that were needed for the Fishtank project. Studies of specific applications designed for these simulators provide insights that relate more directly to the Virtual Fishtank project.

## 5.1. Emergent Behaviors

Much work has been done exploring the interesting behaviors that emerge in systems with many simple interacting agents. A lot of this has come from work at the Media Lab as mentioned earlier ***(Resnick94a) (Resnick94b)***. Some work has been done that deals specifically with simulating the behavior of fish in a biologically accurate way ***(Tu94)***. Also, the behavior of birds in a flock, which is similar to fish in a school, has been studied by a number of people ***(Reynolds87)***. This work suggested what types of inputs that creatures in the Virtual Fishtank would need in order to behave in interesting ways as well as the constraints on their motion that lead to believable movement.

Many ideas about specific behaviors that lead to interesting emergence have been explored using specialized tools. One such tool is Star Logo, which has been in use at the MIT Media Lab for quite some time. Using Star Logo, researchers have explored many emergent phenomena, which will influence the design of creatures in the tank. It shows the types of simple processing that creatures need to be able to do. This is important because the balance between the simplicity and flexibility of behaviors will be vital to the effectiveness of the Fishtank. The balance is explored further in Michael Traver's ***(Travers94)*** work with graphical programming interfaces.

The Swarm simulator ***(Gutowitz93)*** provides a perspective on a wider range of behaviors that those studying emergent systems may wish to explore. This provides an idea of how the Virtual Fishtank simulator may be expanded in the future.

## 5.2. Distributed Simulation on a Network of Workstations

The distributed simulation of Virtual Fishtank presents a unique set of challenges, which do not seem to have been explored together. The Fishtank runs on a network of workstations instead of on a more traditional parallel computer. It is interactive, so it must run in real time, which sets it apart from the majority of the work done in distributed simulation field. Finally, the creatures in the Fishtank interact continuously rather at discreet times as in most simulated systems.

There are many advantages to distributing the Fishtank simulation across a network of workstations ***(Blumofe94)***. The most important of these advantages is that many researchers who study emergent behaviors do not have access to the massively parallel supercomputers that most work in distributed simulation focuses on. There are however, some difficulties involved. The root of all of these is in dealing with latencies between processors that are much higher that the latencies found in parallel computers. The result is that there is a greater focus on limiting interprocessor communication that on finding parallelism within a given problem. The intercommunication problem is often left to the implementer of a specific application rather than being solved in a general way. One such application is Antopia ***(Ebling 89)*** which runs on the Time Warp simulation system ***(Jefferson87)***

Most work on distributed simulation has focussed on systems that interact at discrete points in time, like colliding hockey pucks ***(Beckman88)***, which interact only when two pucks hit. They have used event queues to drive the simulation. The creatures that need to be simulated in the Virtual Fishtank, however, interact as continuously as possible. This means that their behavior must be simulated at each time step of the simulation. This area has not been explored in nearly as much depth as has event driven simulation.

The goal of most simulation systems is to finish the simulation as quickly as possible. Because the Virtual Fishtank is to be rendered in real-time, however, it faces the challenge of running at a constant speed throughout the simulation. This makes many common techniques of distributed simulation unworkable. For example, the Fishtank cannot use rollback, because of its need to continuously render. Also, because there is user interaction with the simulation, it cannot run ahead of real-time. The result is that the simulation must run in constant, synchronized time steps, and so cannot use many of the techniques commonly used by the distributed simulation community.

# 6. Future Work

More work must be done before the Virtual Fishtank museum exhibit is complete. While the distributed octree solves many of these problems, it leaves many issues unaddressed. Many of these issues have been experimented with at various stages of the development of the project, but they have not been fully integrated with the distributed octree architecture. The current implementation allows many different types of creatures with many different types of behaviors to be put into the virtual fishtank. However, it does not allow the creatures or their behaviors to be modified after they are created. At this point, the large-scale movements of the creatures are believable, but the creatures themselves are modeled as rigid bodies without any internal motion. Although the graphical presentation of the fishtank is suitable for testing and demonstration purposes, significant work is needed to reach a level of graphic appeal suitable for the museum exhibit. Finally, the bodies and behaviors of the creatures other than schooling fish need to be created within the framework of the Virtual Fishtank simulation.

## 6.1. Interactive Experimentation in the Virtual Fishtank

In an early, single-computer implementation of the Virtual Fishtank, it was possible for users to modify the behavior of creatures in the tank and to see the results reflected immediately in the creature's behavior. This was a very powerful way to communicate to users the message that the creatures were really following simple local rules rather than complex global procedures. It also made experimentation with behaviors much easier. This was done in two different, but related ways.

The first way of allowing real-time experimentation with the behavior of the creatures in the fishtank was through a simple control that allowed a few important parameters of a behavior to be adjusted. Specifically, the weights of the three rules that lead to schooling could be controlled interactively by the user by moving slider bars. This allowed the user to quickly understand that when, for example, the weight of the rule that had fish move towards the centroid of the school was set very high, the fish would form a swarm rather than a school.

The second, more advanced way that users could experiment with the behaviors of the creatures was through a graphical programming system created by Michael Travers ***(Travers94)***. Users could arrange the blocks in order to specify what conditions creatures would respond to as well as what the creature's response would be. This system allowed almost as much flexibility in specifying behaviors as programming directly in Java, yet it had the potential to be understood by a large number of museum visitors. It could also be used to create a somewhat continuous range of tradeoffs between simplicity and power. This could be done by starting out with most of the graphical components that define an interesting behavior locked in place and allowing the user to change only a few. For example, if blocks representing the three schooling rules were locked in place, while the weights of these rules were adjustable by the user, a simple interface like the one described earlier could be created. Once the museum visitor had gained an understanding of this interface, more and more blocks could be unlocked until the visitor had completely mastered behavior programming in the graphical system.

These interfaces were implemented so that the user could experiment on one computer while the simulation ran on another. The computers would communicate with each other through a network. This was done by creating a system of programming objects in the simulator that corresponded to the blocks in the graphical programming system. These objects could be serialized and sent across the network. These objects could also be named so that an object created at one point in time could be referenced later. By keeping references to the objects that defined the behaviors of creatures in the fishtank, the user interface could change these behaviors and have the results immediately reflected in the creature's behaviors.

Consider, for example, how the schooling example using Michael Traver's graphical programming interface would use the above technique. First, the user interface would construct several fish with a blank behavior and would retain a reference to this behavior. Using the reference to this behavior, constructs called sub-behaviors would then by added to this behavior. The sub-behaviors would represent the three schooling rules and would be made up of other referenced objects like predicates and actions. Three other referenced objects would represent the weights of these sub-behaviors. When the user adjusted the weights through slider bars in the user interface, the references to the weights would be used to modify the weights of the sub-behaviors and so the overall behavior of the fish. As a result, all of the creatures that were created with the schooling behavior would immediately begin to use the new weight when deciding what to do.

The creation of programming objects that could be modified on the computer where the user interface was run and would affect the behavior of creatures on the computer that was simulating the behaviors of the fish was implemented is a straightforward way. The machine with the user interface would ask the simulating machine to create an object of a specific type. The simulating machine would return an identifier that could be used to refer to the object. Then the user interface could then send that identifier along with commands across the network that would modify the object.

In order to make this system work in the distributed fishtank, a method would have to found to distribute this object creation and modification system among all of the computers running the simulation. Ideally, this would be done in an efficient, scalable way. The computer running the user interface would communicate only with the computers that were simulating a region of space that contained a creature that needed the information about the behavior being manipulated. The difficulty in achieving this arises because, as a creature moves around the tank, the computer that needs to know about changes to that creature's behavior. Perhaps, standard approaches for dealing with dynamic, distributed objects could be applied.

## 6.2. Simulation of Bodies

In the current implementation of the Virtual Fishtank, the creatures are modeled as rigid bodies. In order to improve the visual appeal of the museum exhibit, creature's should have moving parts. For example, fish should move their tails when swimming and should use their fins when turning. This type of functionality could be implemented by improving the translation of the desired speed and direction generated by a creature's behavior into the actual movements of the creature.

In the process of doing this, a more realistic model of the actual movement constraints could be implemented as well. The limitations on the direction changes of fish, for example are a substantially simplified model of the motion of fish in the New England Aquarium. A typical reef fish, for example, seems to have two modes of locomotion. One mode is used when the fish does not turn very much, but travels forward rapidly. In this mode, the fish pulls the fins on the sides of it body in and swishes its tail back and forth for propulsion. This is the mode that is modeled in the current implementation of the Virtual Fishtank. A substantially different method of movement is used when a fish is foraging for food or is doing something else that requires a greater degree of maneuverability, but not so much forward speed. In this mode, a fish extends the fins on the sides of its body and uses them to propel itself. Using this technique, a fish can turn approximately seventy-five degrees from side to side almost instantly. After doing this, the fish must pause for a small amount of time while it recovers from this movement. At this point, the fish can make another rapid turn.

In order to implement this type of movement, the procedure that produces a fish's motion given the desired speed and direction generated by its behavior would have to be greatly extended. The determination of which mode of locomotion should be used to achieve the desired movement would have to be made by comparing the weights given to the desired speed to the weights given to the desired direction. The transition between these states would have to be done in a smooth, believable way. Also, in each mode, the appropriate movements for the fish's tail, body, and fins would have to be determined.

A significant obstacle to solving this problem is that it requires a combination of artistic and programming skills. Artistic skills are needed to construct dynamic models of fish that are believable and visually pleasing in every position that the movement procedure may put the fish. Programming skills are needed to create the procedure that figures out how to manipulate this model in a believable way that leads to the desired movements computed by the creature's behavior. These two parts of the problem are tightly coupled, yet there are very few people who have both artistic and programming skills. Therefore, a good solution will require cooperation by people with very different perspectives and understandings of the problem.

## 6.3. Graphic Display

The graphic display of the Virtual Fishtank in its current state is not sufficient for use in the final museum exhibit. One of the biggest problems is the difficulty for the viewer in perceiving the depth of creatures in the tank. Also, the simulator does not currently address the problem of fitting the images that are rendered by different computers together into a consistent display.

The fishtank currently uses two methods of creating a sense of depth for the viewer. First, it uses perspective projection, which makes creatures that are far away from the viewer appear smaller than those that are close to the viewer. It also uses a fog effect to make the colors of creatures that are far away from the viewer shift towards the color of the water in the tank. These two techniques go a long way, but not far enough. Several additional techniques could be used. Static objects could be added to the tank that would obscure distant fish, but not close by ones. For creatures that are close to the bottom of the tank, shadows projected on the floor of the fishtank would create a powerful indication of the how far away each creature was. If these techniques are not effective enough, the distance to the back of the fishtank could simply be decreased, so that most of a creature's interesting movement would be parallel to the view plane and so would be easy for viewers to perceive.

Several difficulties could arise in trying to merge the displays generated by each of the computers simulating the fishtank into one large, coherent display. The synchronization mechanisms used in the simulation of the creature's behavior give each system consistent information about the location and orientation of each creature in the tank. They do not, however, distribute this information to every system that might need to render each creature. Also, because the dimensions of the display generated by the Virtual Fishtank simulation are to be large in comparison to the distance at which museum visitor will view the display, there may be problems with perspective projection.

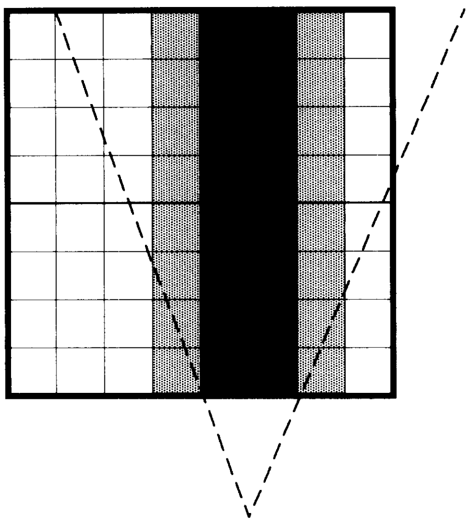

***Figure 6.3-1 – Computers Need Extra Information to Render***

*This is a top view of the main octree as seen by the computer that is simulating the behaviors of creatures in the darkly shaded leaves of the octree. To simulate these behaviors, it receives information about the lightly shaded nodes from the network. In order to render the view shown by the dashed lines, it needs more information about the creatures in additional nodes.*

Situations can arise, where one computer that is simulating a regions of space needs to display creatures that it does not need to know about in order to run the simulation. Figure 6.3-1 illustrates such a situation. In order to render this view correctly, the illustrated system must get information about thirteen additional nodes from other computers on the network. In order to do this, the thirteen nodes could be found before network connections are established. Then, using the same techniques that are used to transmit the information necessary to simulate the behaviors of creatures, the information needed to render correctly could be send across the network. This would increase the simulator's demands on the bandwidth of the network, but this increase seems unavoidable.

Another rendering problem is that, if the rendered outputs of all of the computers are to be projected onto one seamless display, the eye point used for all of these projections must be the same, as shown in Figure 6.3-2. Using a single eye point causes several problems, including strange perspective effects for museum visitors who are not looking at the fishtank from the virtual eye point that is assumed during rendering and poor distribution of the rendering load. The strange perspective effects are an unavoidable consequence of using perspective projection. One possible solution would be to project the

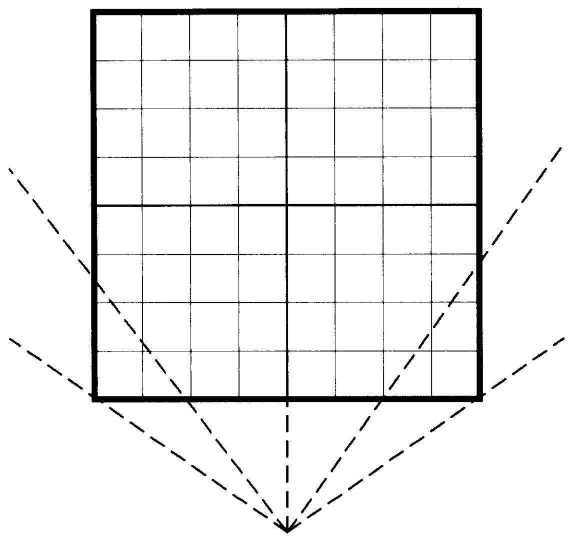

***Figure 6.3-2 – Seamless Perspective Projection from a Single Eye Point***

*This is a top view of the main octree with dashed lines showing four fields of view that must be rendered. In order to render images that will fit together to form a coherent view, each computer running the Virtual Fishtank simulation must use the same eye point. This could cause unacceptable distortion for viewers who are to the left or right of the viewpoint. It also could lead to an imbalance in the rendering load of the computers.*

tank onto the outside of hexagonal prism, using a different eye point for each face of the prism. This way, a museum visitor would be unable to see the renderings that were created using an eye point that was far away from the visitor's actual position. If one face of the cube represented by the octree was mapped around the entire hexagonal prism, perhaps a pleasing display could be created. If this were done, the simulation space could be made to wrap around from left to right which would have the pleasant side effect of having less walls to interfere with interesting creature behavior.

## 6.4. Different Types of Creatures

Perhaps the most important addition to the Virtual Fishtank would be the creation of a greater variety of creatures and behaviors for those creatures. Researchers have discovered many interesting group behaviors that arise from individual creatures following simple rules ***(Resnick94a)***. The Virtual Fishtank provides an excellent framework for experimenting with these behaviors and demonstrating them to museum visitors. If visually appealing bodies that move in realistic ways are created to go along with the behaviors, the experience would be greatly enhanced.

# 7. Conclusion

This thesis explored many of the central problems that must be solved in the creation of the Virtual Fishtank museum exhibit. Specifically, it explored the creation of a distributed octree data structure, which allows efficient computation of visual fields as well as the distribution of the simulation across several computers connected by a network. Also, it dealt with other aspects of the Virtual Fishtank, such as the creation of behaviors and realistic creature motion. The result was a prototype system that demonstrates the principles that will be used in the final implementation of the Virtual Fishtank museum exhibit.

## 8. Bibliography

***Beckman88*** Beckman, B. et al. (1988) Distributed Simulation and Time Warp Part 1: Design of Colliding Pucks In: Distributed Simulation, 1988 Simulation Councils, Inc., San Diego, California, pages 56-60.

***Blumofe94*** Blumofe, D. and D. Park (1994) Scheduling Large-Scale Parallel Computations on Networks of Workstations. In: Third International Symposium on High-Performance Distributed Computing, August 1994, San Francisco, California, pages 96-105.

***Ebling89*** Ebling, M., M. Di Loreto. M. Presley. F. Wieland. and D. Jefferson. (1989) An Ant Foraging Model Implemented on the TimeWarp Operating System." In: Distributed Simulation, 1989 Simulation Councils, Inc., San Diego, California, pages 21-28.

***Gutowitz93*** Gutowitz, H. (1993), A tutorial introduction to Swarm. Technical report, The Santa Fe Institute, 1993. Santa Fe Institute Preprint Series.

***Jefferson87*** Jefferson, D., B. Beckman, F. Wieland, L. Blume, M. Di Loreto, P. Hontalas; P. Laroche, K. Sturdevant; J. Tupman, V. Warren, J. Wedel, H. Younger; and S. Bellenot. (1987). Distributed Simulation of the Time Warp Operating System. In: Proceedings of the Eleventh Annual ACM Symposium on Operating System Principles. ACM, New York, pages 77-93.

***Resnick94a*** Resnick, M. (1994). Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds. MIT Press. Cambridge, MA.

***Resnick94b*** Resnick, M. (1994). Learning About Life. In: Artificial Life Journal, Volume 1, Number 1-2, pages 229-241.

***Reynolds87*** Reynolds, C. W. (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics. SIGGRAPH 1987 Conference Proceedings, pages 25-34.

***Travers94*** Travers, M. (1994) Recursive Interfaces for Reactive Objects, Proceedings of CHI 1994, Boston, Massachusetts, 1994.

***Tu94*** Tu, X. and D. Terzopoulos (1994), Artificial Fishes: Physics, Locomotion, Perception, Behavior. In: ACM Computer Graphics Proceedings, 1994, pages 43-50.